HASH-BASED SIGNATURES

6. SPHINCS+

Alfred Menezes
cryptography101.ca

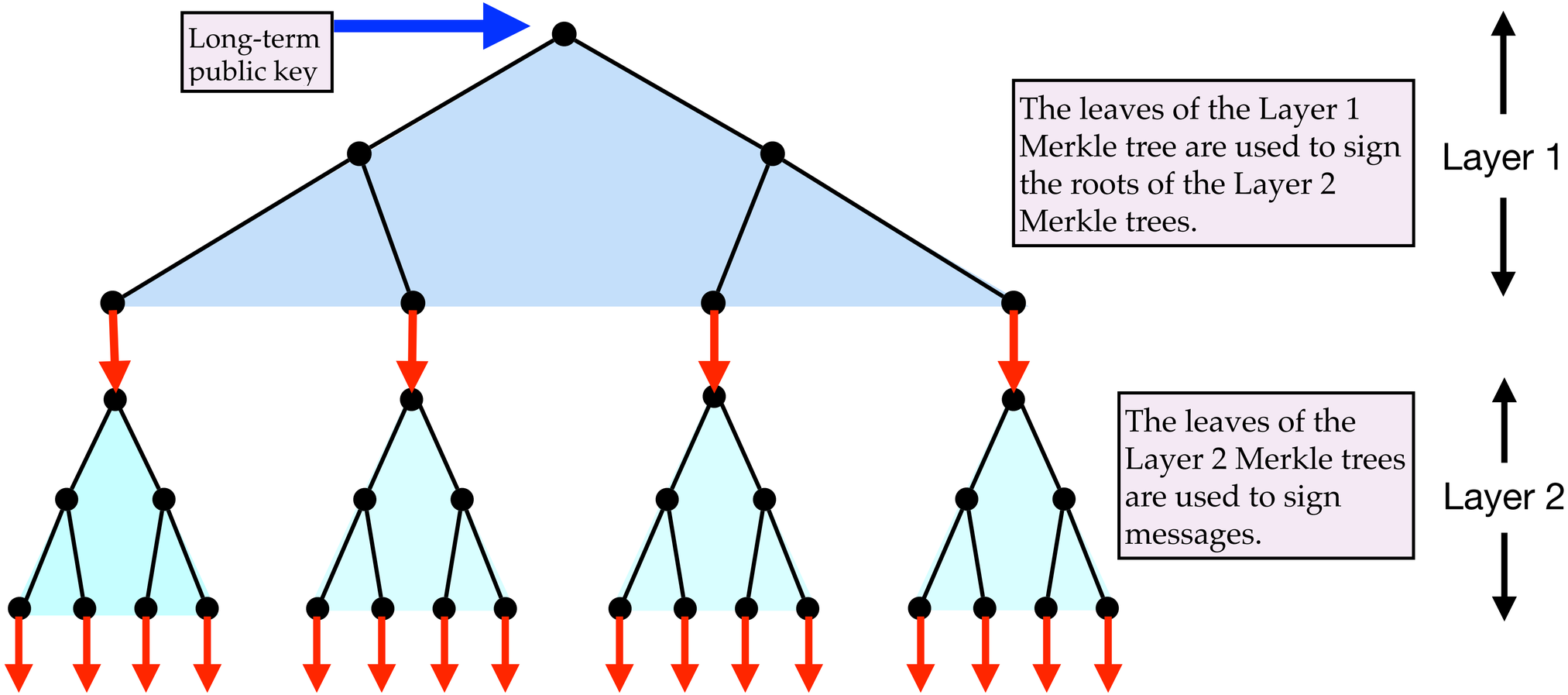
May 2025

Outline

1. Stateful signature schemes
2. SPHINCS+: A stateless signature scheme
3. Few-times signature schemes
4. Virtual hypertrees

Stateful signature schemes

- NIST's SP 800-208: LMS and HSS, XMSS and XMSS^{MT}.



SPHINCS+: A stateless signature scheme

- Designed by Bernstein, Hülsing, Kölbl, Niederhagen, Rijneveld and Schwabe.
- Standardized by NIST in August 2024: **FIPS 205**.
  - Official name is **SLH-DSA** (Stateless Hash-based Digital Signature Algorithm).
- Main idea:
  - Employ a hypertree with a *very large* number of leaves.
  - The leaves represent signature key pairs for an OTS.
  - To sign a message, the signer *randomly* selects one of these key pairs.
  - A limit is placed on the total number of messages that can be signed, ensuring that the probability of key reuse is negligibly small.
- SPHINCS+ employs numerous optimizations. I'll present the two main optimizations:
  - **Few-times signatures** and **virtual hypertrees**.

Few-times signature schemes

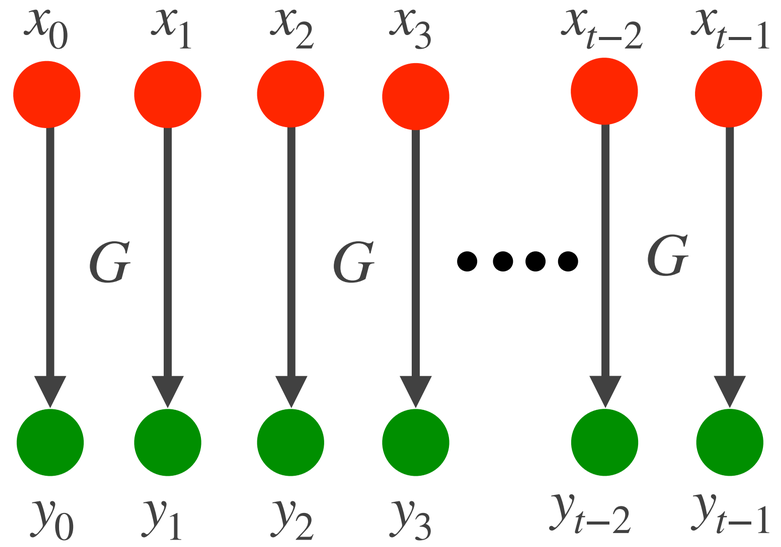
- Unlike a one-time signature scheme, a f**ew-times signature scheme (FTS)** can be used a small number of times (e.g., 5-10 times) before security degrades.
- Main idea: Each signature reveals only a small fraction of the private key components, unlike Lamport's OTS where a single signature reveals half of the private key.
- Example of a few-times signature scheme: **HORS**.

HORS

- **HORS**: Hashing to Obtain Random Subset.
- Proposed by Reyzin and Reyzin in 2002.
- Ingredients: A preimage resistant hash function $G : \{0,1\}^n \rightarrow \{0,1\}^n$ and a second-preimage resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$.
- Parameters: A divisor k of n, $m = n/k$, $t = 2^m$.
- Sample parameters: $n = 256$, $k = 16$, $m = 16$, $t = 2^{16} \approx 65{,}000$.

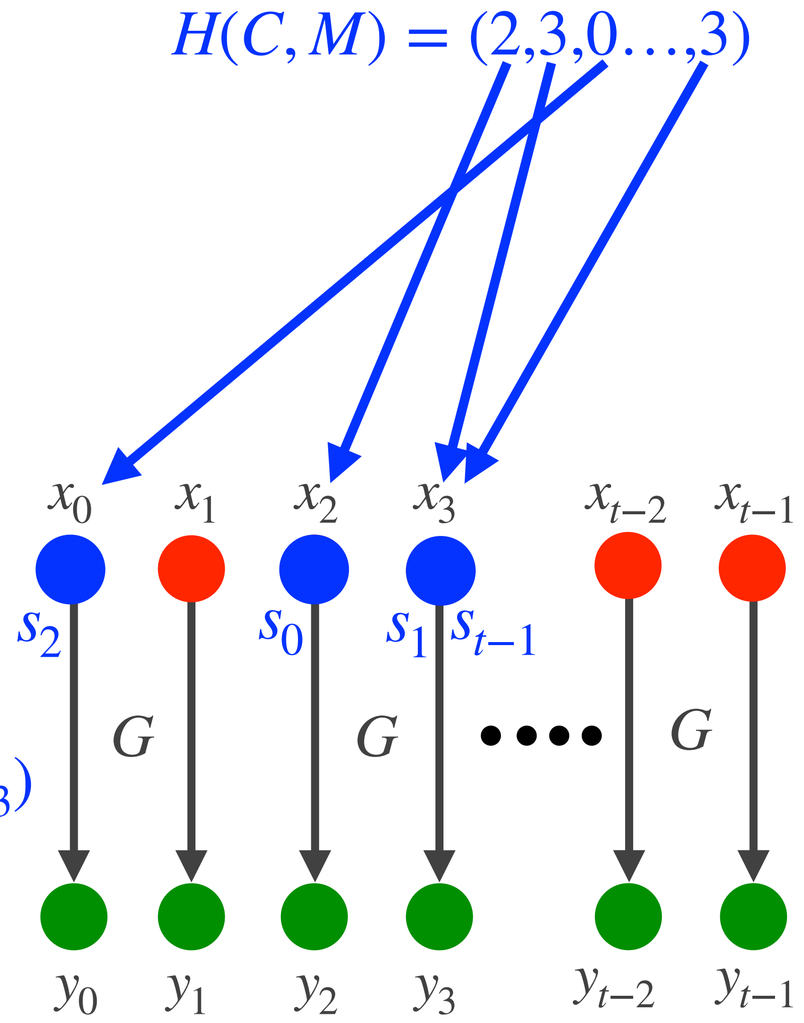
HORS: Key generation

- Alice selects a key pair as follows:
  - Select $x_i \in_R \{0,1\}^n$, for $0 \leq i \leq t-1$.
  - Compute $y_i = G(x_i)$, for $0 \leq i \leq t-1$.
  - Alice's **private key** is $X = (x_0, x_1, \ldots, x_{t-1})$.
  - Her **public key** is $Y = (y_0, y_1, \ldots, y_{t-1})$.



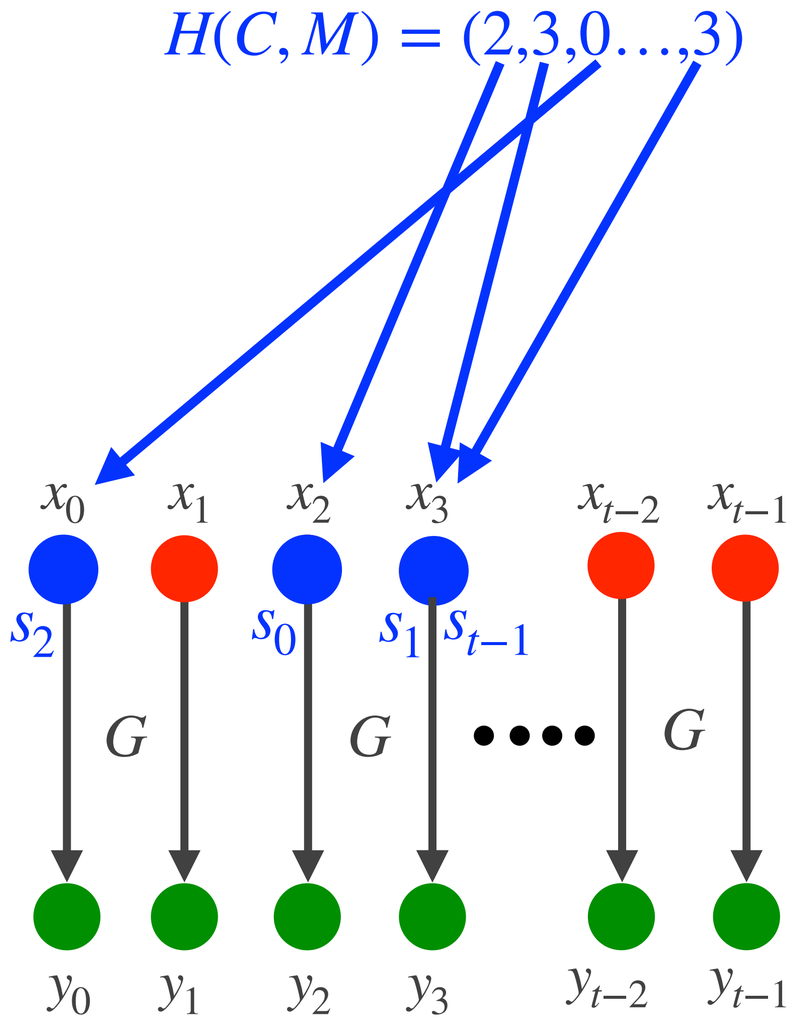
HORS: Signature generation

- To sign a message $M \in \{0,1\}^*$, Alice does the following:
  - Select a message randomizer $C \in_R \{0,1\}^n$.
  - Compute $h = H(C, M)$.
  - Write $h = (h_0, h_1, \ldots, h_{k-1})$, where each $h_i \in \{0,1\}^m$.
  - Set $S = (s_0, s_1, \ldots, s_{k-1})$, where $s_j = x_{h_j}$.
  - Alice's signature on M is (C, S).
- Note: A signature exposes at most k of the t components of X.



HORS: Signature verification

- To verify Alice's signature (C, S) on M, Bob does:
  - Obtain an authentic copy of Alice's public key Y.
  - Compute $h = H(C, M)$.
  - Write $h = (h_0, h_1, \ldots, h_{k-1})$, where each $h_i \in \{0,1\}^m$.
  - Accept iff $G(s_j) = y_{h_j}$ for all $0 \leq j \leq k-1$.



HORS: Security analysis (1)

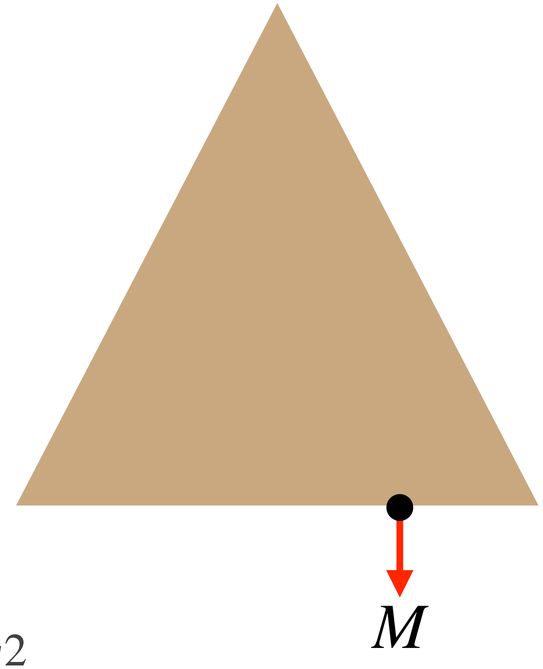
- Suppose the adversary has obtained c signed messages $(M_i, (C_i, S_i))$.
- Since the randomizers C_i were chosen by Alice, the adversary has no control over the hash values $h^{(i)} = H(C_i, M_i)$.
- Assuming that H behaves like a random function, the c signed messages reveal at most ck *randomly-selected* components of X.
- To forge Alice's signature, the adversary needs to find a message M^* and randomizer C^* such that each $x_{h_j^*}$ (where $h^* = H(C^*, M^*) = (h_0^*, h_1^*, \ldots, h_{k-1}^*)$) is among the revealed components of X.
  - This happens with probability at most $(ck/t)^k$.
- If this probability is negligible, Alice can securely sign up to c messages.

HORS: Security analysis (2)

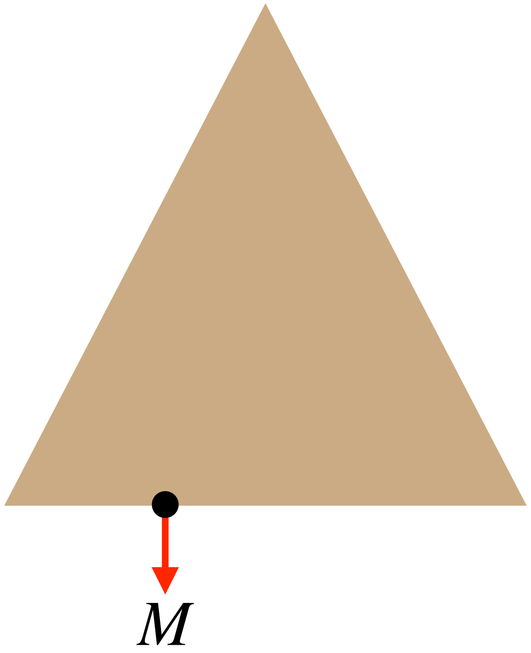
- Example: Suppose $n = 256,\ k = 16,\ m = 16,\ t = 2^{16},$ and $c = 5$.
  - Then the probability defined on the previous slide is $(ck/t)^k \approx 2^{-154}$, which is negligible.
- One drawback: Large public keys (nt bits, or 2.1 Megabytes).
  - Solution: **HORST** (HORS with Trees).
  - The t components of Y are assigned to the leaves of a height-m Merkle tree. The root of the tree is Alice's public key.

Merkle tree

- Consider a height-d Merkle tree whose $N = 2^d$ leaves correspond to the public keys of an OTS.
- To sign a message, a leaf is chosen *at random*.
- Suppose that a limit of S is placed on the number of messages that can be signed.
- By the birthday paradox, the probability that any OTS private key is reused for two or messages is at most $\binom{S}{2}\frac{1}{N} \approx \frac{S^2}{2N}$.
- Example: If $S = 2^{40}$ and $d = 207$, this probability is at most $1/2^{128}$.
- Problem: Constructing a height-207 Merkle tree is completely impractical.



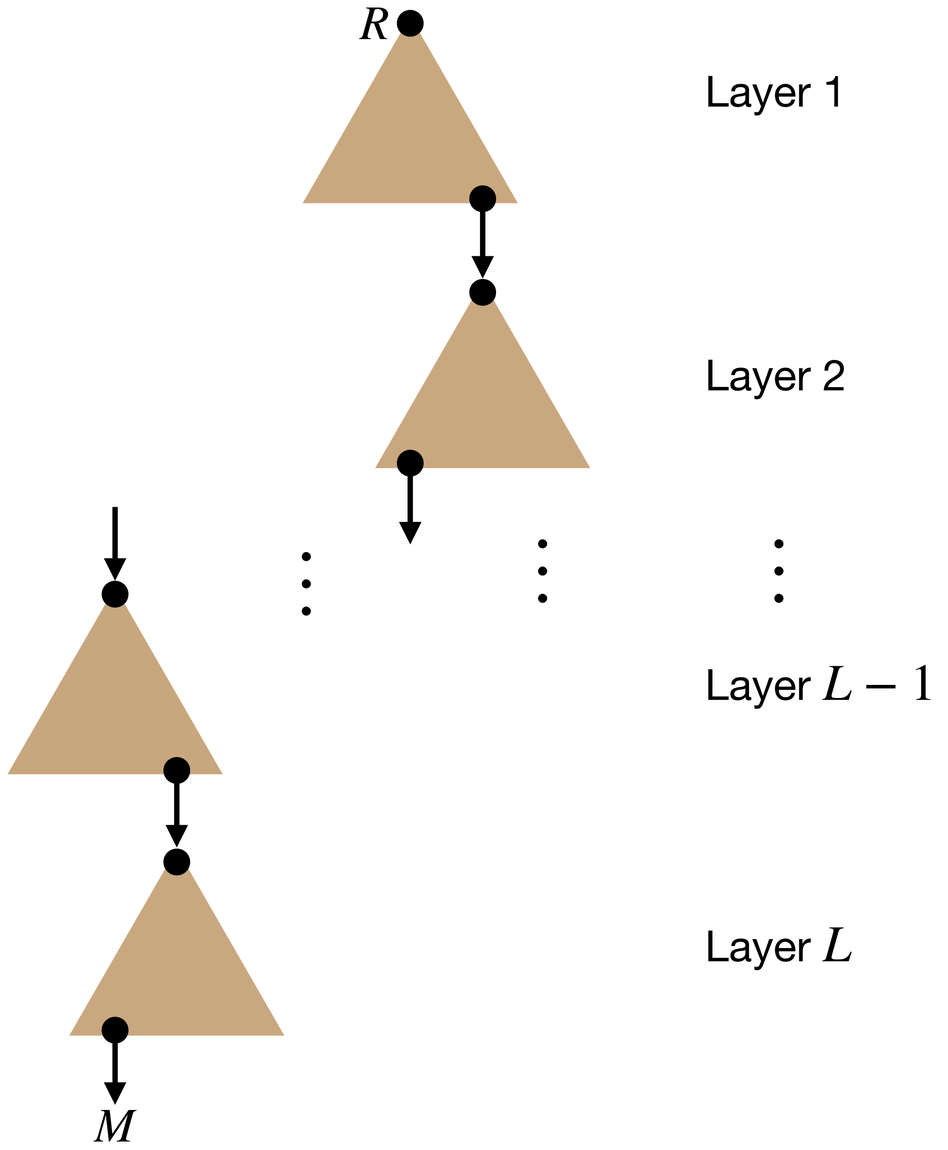
Merkle tree + Few-times signatures



- Each leaf is associated with a key pair for a FTS instead of an OTS.
- Messages are signed using a randomly selected leaf's private key.
- Suppose that a limit of S is placed on the number of messages that can be signed.
- By a generalization of the birthday paradox for multi-collisions, the probability that any FTS private keys is used more than c times is at most $\binom{S}{c}\frac{1}{N^{c-1}} \approx \frac{S^c}{c!\,N^{c-1}}$.
- Example: If $S = 2^{40}$, $c = 5$, and $d = 80$, this probability is at most $1/2^{127}$.
- Problem: Constructing a height-80 Merkle tree is also impractical.

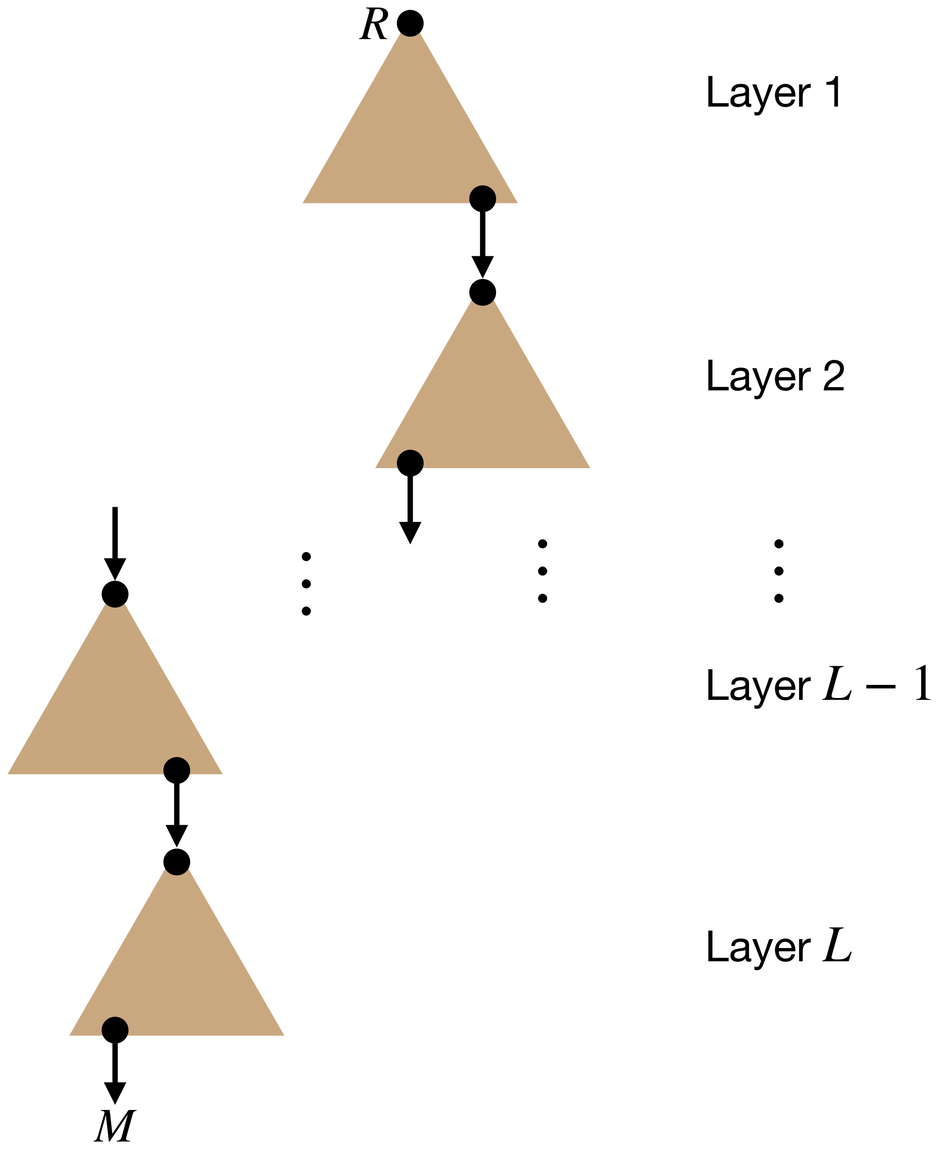
Virtual hypertrees (1)

- A **virtual hypertree** has L layers.
- Each layer consists of height-d Merkle trees.
- Every leaf of the Merkle trees in layers 1 to $L-1$ is associated with an **OTS key pair.**
  - Each OTS private key is used to sign the root of the tree beneath it.
- Every leaf of the Merkle trees in layer L is associated with an **FTS key pair.**
  - The FTS private keys are used to sign messages.
- The root R of the hypertree is the user's **long-term public key**.



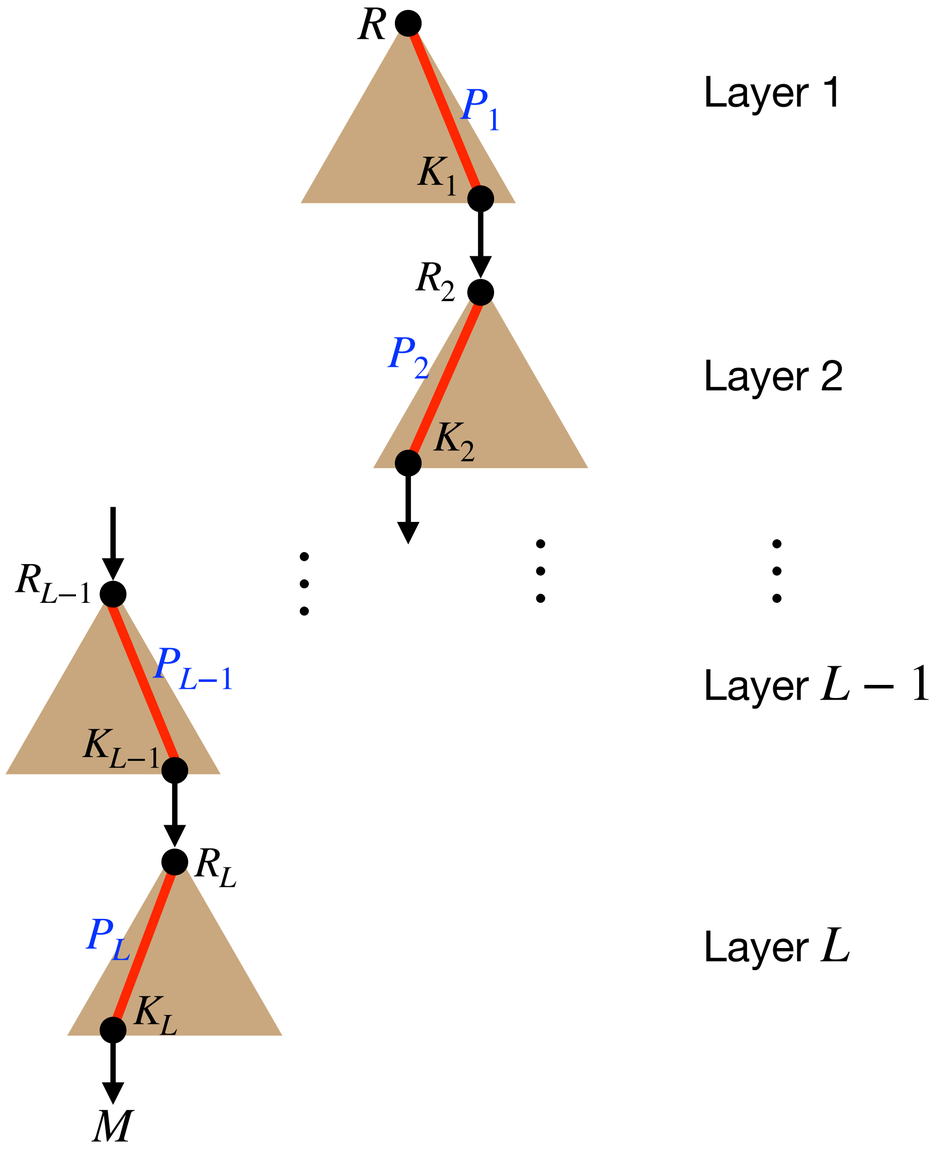
Virtual hypertrees (2)

- All OTS and FTS private keys are derived by applying a prf to a **single secret SEED** and a unique index identifying the corresponding public key.
  - The index might include the layer number, the tree identifier within that layer, and the public key's position within that tree.
- Once SEED is chosen, the entire hypertree is fully defined, even though it might be infeasible to actually construct the entire tree.
  - This is the sense in which the hypertree is "**virtual**".
- Any individual Merkle tree can be efficiently constructed from SEED.



Signature generation

- To sign a message M, Alice selects a random FTS public key K_L and signs the message with the corresponding private key.
- She then derives the root R_L of the Merkle tree containing K_L and derives the authentication path P_L for this key.
- She derives the OTS public key K_{L-1}, signs R_L with the corresponding private key, reconstructs the root R_{L-1} of the Merkle tree containing K_{L-1}, and derives the authentication path P_{L-1} for K_{L-1}.
- This is continued upward until reaching R.
- Alice's signature on M contains the FTS signature, all intermediate OTS signatures, the public keys $K_1, \ldots, K_L$, and the authentication paths $P_1, \ldots, P_L$.



Signature verification

- To verify a signed message, Bob first validates the authentication path P_1 from K_1 to R.
- He then verifies the OTS signature on R_2 using K_1 and validates the authentication path P_2 from K_2 to its Merkle root R_2.
- This is continued downward, layer by layer through the hypertree, until reaching K_L.
- He is then confident of the authenticity of the FTS public key K_L he was sent.
- Finally, Bob verifies the signature on M using K_L.